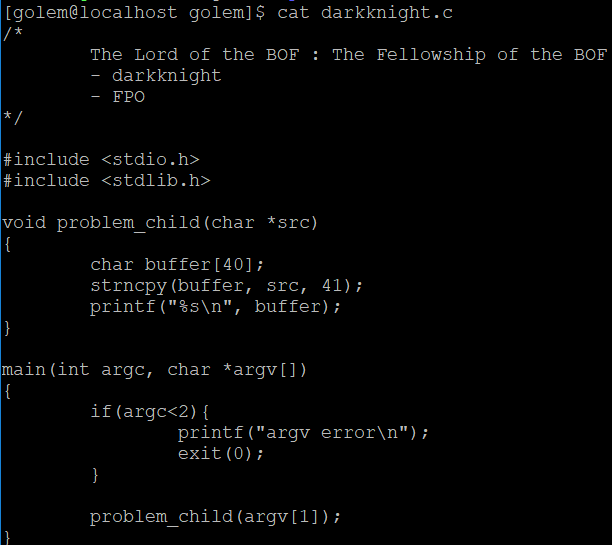
|  |
| --- |
| LOB darkknight |
| 동아리마크  2018. 02. 28 (wed)  Dong-a .univ  COLONY  Young jin kim |

목차

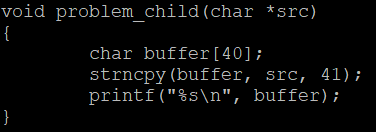
1. 소스코드 이해
2. 소스코드 분석
3. 공격
4. **소스코드 이해**



[ golem.c의 소스코드 ]

|  |  |
| --- | --- |
| Problem\_child함수 정의  Char형 buffer 40byte 선언  Src에서 41byte만큼 buffer에 복사  Buffer 출력 | main함수 정의  인자값이 1개 이하이면 프로그램 종료  Problem\_child 함수 실행  FPO(Frame Pointer Overfow) : sfp 조작기법  -FPO를 통해 원하는곳으로 Return이 가능하다 |

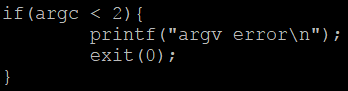
1. **소스코드 분석**



[ 조건 1 ]

첫번째, src를 buffer로 복사하는 과정에서 1byte가 남는다.

두번째, 1byte는 sfp에 해당하는 영역을 조작해서 FPO공격이 가능하다.



[ 조건 2 ]

첫번째, 인자는 2개 이상 사용한다

**FPO(Frame Point Overflow)에 대해 알아보자**

위의 darkknight.c 를 기반으로 설명을 해보겠다.

프로그램이 실행되면 main()안에서 problem\_child()함수 실행

< main frame > => <problem\_child>

[sfp][ret] => [buffer(40)][sfp][ret]

=====================================스택의 구조이다

< problem\_child> => < main frame>

[buffer(40)][sfp][ret] => [sfp][ret]

^-esp ^-ebp

=====================================leave, ret 실행

Leave = mov ebp esp, pop ebp ret = main의 코드 메모리값으로 돌아감

< problem\_child > => < main frame >

[buffer(40)][sfp][ret] => [sfp][ret]

^-esp ^-ebp

여기서 pop ebp가 되면 esp+4가 되면서 ret가 실행 되어 다시 메인함수가 돌아간다,

=========================================정상적인 프로그램 실행 구조

< problem\_child > => < main frame >

[buffer(40)][sfp][ret] => [sfp][ret]

^-esp

=========================================ebp변조 하였을 때 메모리

sfp값을 변조한다 -> ebp값이 변조된다 -> mov로 인해 ebp의 위치에 esp가 간다

->프로그램은 정상적으로 ret되어 실행되고 메인함수가 끝난다. -> 메인함수가 끝날 때

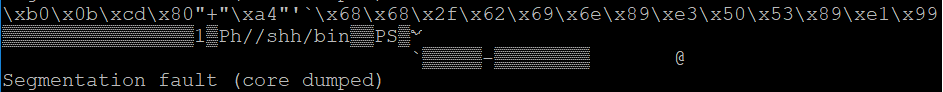
Leave,ret 어셈블리어가 실행되는데 이때 변조된 ebp의 위치에 esp가 가게 되고

Esp+4가 되면서 명렁어가 실행되는데 이때 FPO가 일어난다.

결론 : 쉘코드를 넣어넣고 그 주소값을 찾아서+4한 메모리값으로 ebp를 넣으면 공격이 된다.

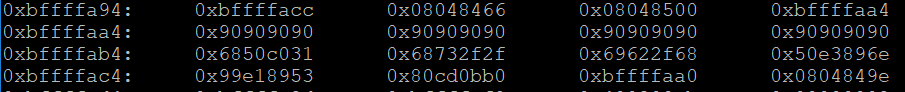
+) ebp 뒤에 ret 주소도 추가해 주어야한다.

**2.소스코드 분석**



[ 코어파일 생성 ]

첫번째, 세폴을 내서 코어파일을 생성한다.

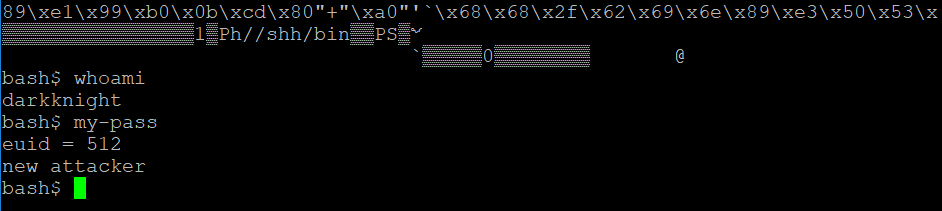


[ 메모리 분석 ]

첫번째, 버퍼가 시작하는 메모리값 : 0xbffffa4

두번째, sfp수정한 메모리값 : 0xbffffac(값 : bffffaa0)

**3.공격**



첫번째, 페이로드의 구성은 다음과 같다

|  |  |  |  |
| --- | --- | --- | --- |
| Buffer[40] | | | Sfp[1] |
| (Ret)\xbffffaa8 | “\x90”\*12 | ShellCode[24] | “\xa0” |

두번째, 익스플로잇!